VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

POSILOVANÉ UČENÍ PRO HRANÍ HRY STARCRAFT

REINFORCEMENT LEARNING FOR STARCRAFT GAME PLAYING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **LUKÁŠ CHÁBEK**
AUTHOR

VEDOUCÍ PRÁCE **Doc. RNDr. PAVEL SMRŽ, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/19963/2017/xchabe00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Chábek Lukáš**
Obor: Informační technologie
Téma: **Posilované učení pro hraní hry Starcraft**
Reinforcement Learning for Starcraft Game Playing
Kategorie: Umělá inteligence

Pokyny:

1. Seznamte se s rozhraními, zpřístupňujícími hru Starcraft programům.
2. Zpracujte přehled metod pro hraní strategických her a zlepšování strategií pomocí technik z oblasti posilovaného učení.
3. Implementujte systém s pokročilou herní strategií a nezbytná rozhraní pro zařazení do soutěže.
4. Vyhodnoťte vytvořený systém srovnáním s ostatními účastníky.
5. Vytvořte stručný plakát prezentující práci, její cíle a výsledky.

Literatura:

- dle doporučení vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- funkční prototyp řešení

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrž Pavel, doc. RNDr., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Práce se zabývá metodami strojového učení aplikovanými pro hraní strategických her v realném čase. V práci se zabývám metodou strojového učení Q-learning založenou na zpětnovazebním učení. Praktická část práce je implementování agenta pro hraní hry Starcraft II. Mnou navržené řešení se učí spolupráci 4 jednoduchých sítí, které se nadále učí optimálně provádět jim přístupné akce ve hře. Analýza a vyhodnocení systému jsou provedeny experimentováním a sbíráním statistik z odehraných her.

Abstract

This work focuses on methods of machine learning for playing real-time strategy games. The thesis applies mainly methods of Q-learning based on reinforcement learning. The practical part of this work is implementing an agent for playing Starcraft II. Mine solution is based on 4 simple networks, that are colaborating together. Each of the network also teaches itself how to process all given actions optimally. Analysis of the system is based on experiments and statistics from played games.

Klíčová slova

Strojové učení, Umělá Intelience, Starcraft II, RTS, Q-Learning, Deep-Q-Learning, PySC2

Keywords

Machine Learnig, Artificial Intelligence, Stracraft II, RTS, Q-Learning, Deep-Q-Learning, PySC2

Citace

CHÁBEK, Lukáš. *Posilované učení pro hraní hry Starcraft*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. RNDr. Pavel Smrž, Ph.D.

Posilované učení pro hraní hry Starcraft

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Pavla Smrže, doc. RNDr., Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Lukáš Chábek
17. května 2018

Poděkování

Moje poděkování patří vedoucímu bakalářské práce dor. RNDr. Pavlovi Smržovi, Ph. D. za jeho trpělivost a odbornou pomoc.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá konceptem strojového učení pro hraní strategických her, ve kterých probíhají veškeré akce v reálném čase. Tyto hry se nejčastěji nazývají pojmem *Real-Time Strategy*, ve zkratce *RTS*. Tento žánr her vznikl v 80. let 20. století titulem *Herzog Zwei*, která byla první *RTS* strategie vytvořená firmou *Techsoft* a publikována velice známou herní společností *Sega*. Tímto byl odstartován velký zájem o hry tohoto žánru a následně vznikalo plno dalších titulů.

Jedním z těchto titulů je také *Starcraft* od společnosti *Blizzard Entertainment*, který nejen že nabídl v té době jeden z nejlepších herních ovládání, ale také oslovil hráče svým velice zajímavým a chytlavým příbehem. Oblíbenost této hry byla tak velká, že společnost neotálela a v ten samý rok vydala i pokračování tohoto oblíbeného titulu nesoucí název *Starcraft: Brood War*.

V roce 2010 byl vydán první díl trilogie *Starcraft II* nesoucí název *Starcraft II: Wings of Liberty*, která obsahovala pokračování příbehu zaměřeného na jednu z hlavních ras ve hře, a to rasu *Terran*, která představuje obyčejné lidi. Druhým dílem, vydaným v roce 2013, byl *Starcraft II: Heart of the Swarm*, který byl zaměřen na druhou rasu ze hry a to *Zerg*, představující jakysi organismus, který není velice technicky vyspělý, ale má svoji sílu v počtu a jednotě. V posledním díle vydaném v roce 2015 a nesoucí název *Starcraft II: Legacy of the Void* je hra zaměřena na poslední rasu *Protoss*. Tato rasa je specifická svou technologickou vyspělostí a sílou každého jedince.

1.1 Motivace

Starcraft jako takový je velice komplexní co se týče jak ovládání, tak pozorování. Již při vydání hry bylo plno fanoušků, kteří hráli a učili se všechny možné herní techniky a strategie, kterými by mohli dominovat ve hře. Ovšem *Starcraft* byl jednou z her, které byly mnohem těžší, než jiné tituly. V porovnání například s *Age of Empires*, kde všechno je poznatelně pomalejší, *Starcraft* vyčníval svojí rychlostí a možností porazit protihráče již v prvních pěti minutách.

První verze *Starcraftu* byla velice těžká i co se týče ovládání. V jedné výběrové skupině mohlo být maximálně 12 jednotek, přičemž maximální počet pro jednoho hráče bylo 200. To velice komplikovalo celé ovládání hry a hlavně tu nejdůležitější část *Micro*[1].

[1]Ovládání jednotek v boji, uhýbání a správné rozmísťování, které vede k co možná největšímu poškození protihráče, nebo naopak nejmenšímu poškození vlastních jednotek.

Obrázek 1.1: Starcraft 1

Jedna z nejpopulárnějších taktik *Zerg Rush* vyžadující postavit pouze jednu budovu, která poté dovoluje vytvářet velké skupiny jednotek. Tyto jednotky nejsou sice silné, ale pokud se objeví ve velkém množství v brzkých minutách a nejsou udělána patřičná opatření, hra je u konce. V druhé verzi hry *Starcraft II* je vše trochu promícháno. Není omezené množství jednotek, které mohou být označeny a manipulovány, čímž se zjednoduší například přikázání všem útočným jednotkám, aby se přesunuly do základny nepřítele. Ve *Starcraftu II* je vše poněkud rychlejší. Budovy se staví rychleji a oblíbenou taktiku *Zerg Rush* je možné provést i v této nové verzi.

Obrázek 1.2: Starcraft II

Každá jedna jednotka má definované, proti kterým jednotkám je silná a naopak. Pokud dva hráči proti sobě pošlou své armády, s nějvětší pravděpodobností vyhraje ten hráč, který má více jednotek, které jsou zaměřeny na daný typ jednotek. Pokud oba hráči začnou své útočné jednotky ovládat a například dávat jim povel, aby všichni zaměřili jeden cíl, hovoříme o *Micro*. Například obrana proti již zmiňované taktice *Zerg Rush* je u každé rasy specifická. Pokud je dobře za danou rasu zahráno, může jediná jednotka odrazit celý tento ničivý útok.

Další důležitou částí ovládání hry je tzv. *Macro*[2]. Pokud ani jeden z hráčů neplánuje nějakou strategii, která by měla za následek konec hry do pár minut, je většinou výherce ten, kdo nejlépe provede zmiňované *Macro*. Správné načasování stavění a vytváření jednotek

[2]Ovládání a stavění budov a vylepšení, rozšiřování svých hlavních budov a vytváření jednotek.

je taktéž jednou z klíčových vlastností hry. Pokud hráč rozšiřuje svoji základnu a stanoví optimální přísun surovin, bude později, pokud například zaútočí a příjde o své jednotky, schopen obnovit celou svoji armádu a jednoduše čelit protiútoku.

Všechna tato komplexnost a nutná přesnost je velice zajímavým úkolem pro návrh a implementaci agenta, který se bude učit vykonávat a zdokonalovat nějaké tyto techniky.

1.2 Stanovení cílů a struktura práce

Hlavním cílem této bakalářské práce je vytvořit agenta pro hru *Starcraft II*. Pro tvorbu agenta je použita metoda zpětnovazebního učení nazývaná Q-learning. Jelikož je zpřístupnění vytváření takovýchto agentů ve hře *Starcraft II* velice nové, není ještě vše dokonale odladěné a nástroje, které jsou již k dispozici v repozitářích, jsou stále ve vývoji. Mým cílem je vytvořit agenta, který dokáže sám odehrát zápas na mnou vybrané mapě za mnou zvolenou rasu a vyhrát. Jelikož je vše ve vývoji, není zatím možné hrát proti vytvořenému agentovi nebo spustit dva agenty proti sobě. Tato funkce je již ve vývoji a není zcela dokončená.

Můj hlavní cíl lze rozdělit do několika podcílů:

- Podcíl 1 - definování prostředí hry Starcraft II a přístupu, jakým bude agent operovat s hrou
- Podcíl 2 - definování statických instrukcí, které agent nebude provádět, ale budou se provádět samy
- Podcíl 3 - definování instrukcí, které agent bude provádět a zároveň určení části pozorování a metod, podle kterých se bude agent rozhodovat
- Podcíl 4 - Vybrání frameworku pro vývoj agenta
- Podcíl 5 - Výběr verze hry a synchronizace agenta s vybranou verzí
- Podcíl 6 - definování funkcí pro pozorování, které budou zaznamenávat dění ve hře a zprostředkovávat zpětnou vazbu pro agenta
- Podcíl 7 - definování funkcí a instrukcí pro agenta, ze kterých bude vybírat a které budou připraveny pro správné zpracování těchto instrukcí

Kapitola 2

Rozbor řešené problematiky

”Umělá inteligence je věda o vytváření strojů nebo systémů, které budou při řešení určitého úkolu užívat takového postupu, který kdyby ho dělal člověk bychom považovali za projev jeho inteligence.” [1]

Umělá inteligence, o které se bavíme v herním průmyslu, je jakýsi model předprogramovaného chování nějaké entity, která má za cíl provádět postupně dané příkazy a podle nějakých jednoduchých pravidel upravovat jejich posloupnost. Nejčastěji se tyto předprogramované modely vytvářejí s jakýmsi hendikepem, aby měl hráč šanci vyhrát. Například ve strategiích, kde je možné zvolit postup, který pokud se provede bez chyb a se správným načasováním je skoro nepřemožitelný.

Ovšem umělou inteligencí, kterou se zabývá tato práce, je taková, která má v sobě prvky učení a rozhodování podle pozorování. Tato inteligence obsahuje algoritmus, který má jakousi možnost pozorování prostředí a stavů, a podle toho se rozhoduje, jakou akci provede.

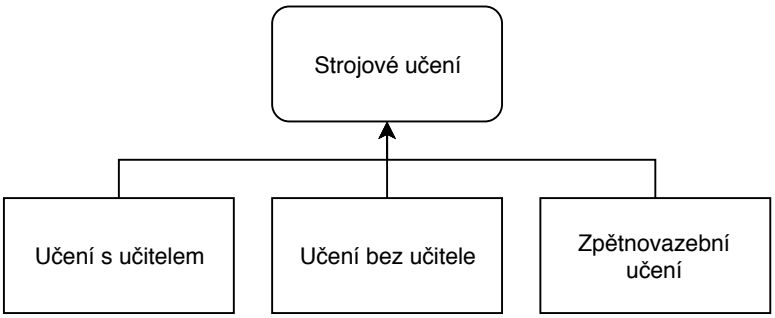
2.1 Strojové učení

Strojové učení je metoda pro vytváření umělé inteligence. Tímto učením jsou myšleny algoritmy, které simulují reálné učení. Nejprve je vytvořen model, který obsahuje parametry, podle kterých se rozhoduje, jaký bude výstup tohoto modelu. Pomocí algoritmů strojového učení a různých matematických metod jsou parametry tohoto modelu upravovány tak, aby se docílilo co největší možné přesnosti výstupu s porovnáním od již známé správné hodnoty.

Strojové učení se dnes používá ve velmi širokém spektru odvětví. Například vyhledávač *Youtube* má v sobě implementaci strojového učení, které navrhuje jaká videa by mohla uživatele zajímat. Dalším příkladem mohou být titulky k videím na zmiňované platformě. Pokud nejsou k dispozici titulky a agent rozpozná jazyk, nabídne titulky, které se snaží vygenerovat pomocí záznamů hlasu. Agent také rozpoznává více osob, a obarvuje titulky různými stupni šedi. [5] [11] [13]

Strojové učení se také využívá v problémech, kde není možné projít všechny stavy problému. Pod tímto si můžeme představit například hru *Go*, což je desková hra pro dva hráče, která má původ v Číně. Hra má několik jednoduchých pravidel, ale komplexnost této hry je natolik složitá, že není možné projít všechny stavy, do kterých se hra může dostat. Pro tuto hru byl vytvořen společností *Google DeepMind* agent nesoucí název *AlphaGo*, který měl za úkol naučit se hrát hru *Go* a porazit světového šampióna.

Metody strojového učení můžeme klasifikovat podle metody učení. Pokud agent má či nemá k dispozici správné hodnoty, které by měl dávat na výstup, rozlišujeme učení s učitelem, učení bez učitele a posilované učení.



Obrázek 2.1: Rozdělení strojového učení

2.2 Učení s učitelem

Učení s učitelem spočívá v trénování agenta na správných výsledcích. Pro daný problém jsou vstupní proměnné X a nějaká výstupní proměnná Y. Pomocí algoritmů se agent učí najít mapovací funkci, která převede vstupy X na výstupy Y 2.1.

$$Y = f(X) \tag{2.1}$$

Cílem je aproximovat mapovací funkci tak, aby při nových vstupních datech bylo možné předpovědět výstupní proměnnou pro tato data.

Tento proces se nazývá učení s učitelem, protože na učení z trénovacích dat může být nahlíženo jako na učení s učitelem, kde algoritmus má k dispozici trénovací sadu dat a postupně vybírá odpovědi, které si myslí, že jsou správné. Pokud jeho odpověď není správná, je opravena správnou odpovědí. Toto učení je dokončeno, když algoritmus dosáhne přijatelné úrovně úspěšnosti. [4] [10]

Učení s učitelem může být dále rozděleno do skupin a to do Regrese a Klasifikace.

- **Regrese**: Problém regrese je když výstupní proměnná je nějaká reálná hodnota jako například váha, nebo cena.
- **Klasifikace**: Problém klasifikace je když výstupní proměnná je nějaká kategorie jako například barvy červená a modrá, nebo zdravotní stav "nemocný"a "zdravý"

2.3 Učení bez učitele

Učení bez učitele je takové učení, kde jsou nějaká vstupní data, ale neexistují korespondující výstupní data. Cílem učení bez učitele namodelování základní struktury, nebo rozdělení dat aby bylo možné se naučít více o daných datech.

Toto učení se nazývá učení bez učitele, jelikož oproti učení s učitelem, zde nejsou žádná data, z kterých by bylo možné ověřit správnost výstupů. Algoritmy proto musí samy objevit strukturu dat. [4] [6] [7]

Učení bez učitele může být dále rozděleno do skupin a to do Klastrování a Sdružování

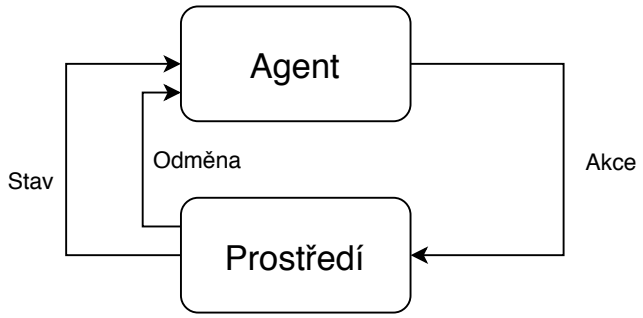
- **Klastrování**: Problém klastrování je, když se hledají společné vlastnosti. Například společný zájem zákazníků.
- **Sdružování**: Problém sdružování je, když se hledají pravidla pro popis velké skupiny dat, například lidé, kteří kupují X většinou koupí i Y.

Mezi populární algoritmy pro učení bez učitele jsou:

- k-means pro klastrování
- Apriori algoritmus pro sdružování

2.4 Zpětnovazební učení

Zpětnovazební učení nebo také posilované učení je metoda strojového učení, která se učí z určitě situace vyvodit akci, která maximalizuje odměnu nebo vyprodukuje určitý zpětnovazební signál. Agent neví, které akce má zrovna použít, jako v ostatních formách strojového učení, ale místo toho musí sám zjistit, jaké akce vrátí nejvyšší možnou odměnu. V těch nejzajímavějších a nejvyzívavějších případech, akce nemusí ovlivnit pouze okamžitou odměnu, ale mohou ovlivnit, jaká bude další situace a tím pádem také všechny následující odměny. Dvě charakteristiky, *Pokus a Omyl* a *Vyhledávání a opoždění odměny* jsou dvě nejdůležitější rozlišovací vlastnoti zpětnovazebního učení. [14]



Obrázek 2.2: Zpětnovazební učení

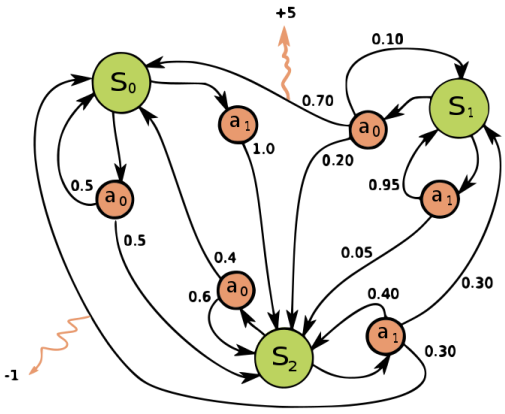
Zpětnovazební učení se využívá i v problémech, kde není vůbec známo, jestli daná akce byla správná nebo špatná a výsledek je znát až na úplném konci. [15]

2.4.1 Markovské rozhodovací procesy

Markovské rozhodovací procesy poskytují matematický model pro modelování rozhodování v situacích, kde výstupy jsou z části náhodné a z částí pod kontrolou rozhodování. Markovské rozhodovací procesy jsou užitečné pro studium širokého spektra optimalizačních problémů řešitelných přes dynamické programování a zpětnovazební učení.

Markovský rozhodovací proces je pětice $(S, A, P_{\cdot}(.,.), R_{\cdot}(.,.), \gamma)$ kde

- S je konečná množina stavů
- A je konečná množina akcí



Obrázek 2.3: Příklad jednoduchého Markovova rozhodovacího procesu se třemi stavy a dvěma akcemi
Zdroj: https://en.wikipedia.org/wiki/Markov_decision_process

- $P_a(s, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ je pravděpodobnost že akce a ve stavu s v čase t povede ke stavu s' v čase $t + 1$
- $R_a(s, s')$ je okamžitá odměna (nebo očekávaná odměna) po přechodu ze stavu s do stavu s', kvůli akci a
- $\gamma \in [0, 1]$ je faktor, který reprezentuje rozdíl v důležitosti mezi budoucí a současnou odměnou.

Problémem Markovských rozhodovacích procesů je najít strategii pro rozhodování. Funkce π která specifikuje akci $\pi(s)$ kterou se rozhodne když nastave stav s. Cílem je vybrat stragegii, která maximalizuje kumulativní funkce náhodné odměny, typicky očekávanou diskontovanou sumu za potenciálně nekonečnou dobu:

$$\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) \quad \text{(kde zvolíme } a_t = \pi(s_t)) \tag{2.2}$$

kde π je diskontní faktor splňující $0 < \pi \leq 1$. Typicky se blíží 1. Díky Markovské vlastnosti může být optimální strategie pro tento speciální problém zapsán jen jako funkce s, jak je předpokládáno výše.[2][9]

2.4.2 Q-Learning

Q-learning je forma zpětnovazebního učení 2.4, která nepotřebuje model prostředí ve kterém je použité. Q-learning algoritmus je založen na hodnotách stav-akce dojic. Hodnota $Q(s, a)$ je definována jako očekávaný diskontovaný součet budoucích odměn získaných akcemí a ze stavu s a následováním optimální strategie. Jakmile jsou všechny tyto hodnoty naučeny, jsou vybírány takové akce, které představují pro daný stav nejvyšší Q-hodnotu.

Po inicializaci libovolnými hodnotami jsou Q-hodnoty určovány na základě získaných zkušeností následovně:

1. Ze současného stavu s vyber akci a. Tímto se získá odměna r s přechodem do dalšího stavu s'.

2. Aktualizace hodnoty $Q(s,a)$ založená na získané zkušenosti následovně:

   malá změna v $Q(s,a) = x[r + y * maxQ(s',b) - Q(s,a)]$, kde x je míra učení a $0 < y < 1$ je faktor stárnutí

3. Jdi na krok 1.

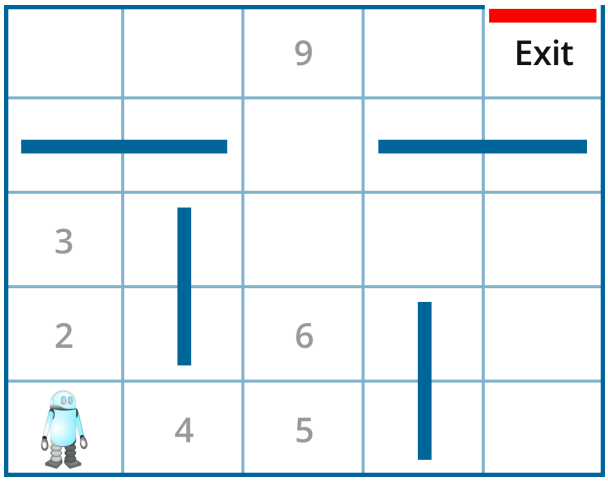
Tento algoritmus má garantované konvergování k správným Q-hodnotám s pravděpodobností 1, pokud je prostředí, ve kterém je použit stacionární a závislé na současném stavu a provedené akci v něm. Hodnoty $Q(s,a)$ neboli Q-hodnoty jsou uloženy v tabulce, kde každá dvojice stav-akce je průběžně navštěvována, a míra učení je postupně s časem snižována.[3][16]

Řekněmě, že máme příklad, kde robot má projít bludištěm, které je znázorněno na obrázku 2.4. Na začátku jsou inicializované Q-hodnoty na 0 pro každou stav-akce dvojici. Přesněji $Q(s,a) = 0$ pro všechny stavy s a akce a. Tímto v podstatě říkáme, že nemáme žádné informace na dlouhodobou odměnu pro každou stav-akce dvojici.

Když agent započne učení, volí akci a ve stavu s a dostává odměnu r. Dále zjistí, že se změnil stav na nový s'. Agent aktualizuje $Q(s,a)$ pomocí následující formule:

$$Q(s,a) = (1 - mira\_uceni) * Q(s,a) + mira\_uceni(r + faktor\_starnuti * maxQ(s',a)) \tag{2.3}$$

Míra učení je číslo mezi 0 a 1. Je to váha pro novou informaci oproti staré informaci. Nová dlouhodobá odměna je současná odměna r sečtená se všemi budoucími odměnami v nadcházenícím stavu s' a pozdějších stavech, za předpokladu, že agent v budoucnu vždy vybere nejlépe ohodnocenou akci. Budoucí odměny jsou ovlivněny faktorem stárnutí, což je taktéž hodnota mezi 0 a 1, což znamená, že budoucí odměny nejsou tak hodnotné, jako současné odměny.



Obrázek 2.4: Robot učící se cestu skrz bludiště.

V těchto úpravách je tato Q paměť přenášena do všech budoucích kroků. Jak agent prochází všechny možné stavy a zkouší všechny různé akce, nakonec se naučí optimální Q-hodnoty pro všechny stav-akce páry. Poté může vybírat akci v každém jednom stavu, které je dlouhodobě optimální.

Pokud vezmeme v úvahu Obrázek 2.4, robot začíná v levém dolním rohu. Každé políčko má svoje číslo označující stav. Robot má na výber ze čtyř akcí (Nahoru, Dolu, Vlevo, Vpravo), ale některé stavy jsou limitovány, například stav 1 (počáteční stav) má pouze možné akce Nahoru a Vpravo. Když robot narazí na zeď, dostane odměnu -1, pokud se robot přesune do otevřeného prostoru, odměna je 0 a pokud se dostane do cíle, obdrží odměnu 100. Ovšem jednorázová odměna je odlišná od Q-hodnot. V podstatě máme:

$$Q(4, vlevo) = 0.8 * 0 + 0.2 * (0 + 0.9 * Q(1, vpravo)) \tag{2.4}$$

$$Q(4, vpravo) = 0.8 * 0 + 0.2 * (0 + 0.9 * Q(5, nahoru)) \tag{2.5}$$

Kde míra učení je 0.2 a faktor stárnutí je 0.9. Nejhodnotnější akce ve stavu 1 je vpravo a poté ve stavu 5 je nejlepší akce nahoru. Q(1, vpravo) a Q(5, nahoru) mají různé hodnoty, protože ze stavu 1 je delší cesta k cíli než ze stavu 5. Jelikož faktor stárnutí ovlivňuje budoucí odměny, odečítáme přidané kroky k dosažení cíle. Tím pádem Q(5, nahoru) má vyšší hodnotu než Q(1, vpravo). Kvůli tomu Q(4, vpravo) má vyšší hodnotu než Q(4, vlevo) a tím pádem nejlepší akce ve stavu 4 je vpravo.

Q-learning vyžaduje, aby agent prošel co možná nejvíckrát stav-akce dvojce. Pouze poté má agent kompletní představu o tom, jak vypadá svět, ve kterém se nachází (v tomto případě bludiště). Q-hodnoty reprezentují optimální hodnoty při výběru nejlepší sekvence akcí. Tato sekvence akcí se také nazývá strategie nebo politika. [8]

2.5 Starcraft II

Starcraft II je pro strojové učení prostředí, ve kterém nelze vybrat jedna stragtegie, která by vždy zajistila vítězství. Hra obsahuje spoustu faktorů, které mohou ovlivnit výsledek střetu dvou armád. Nejdůležitějším faktorem je poměr ekonomika / vojsko, kde je potřeba zajistit silnou ekonomiku rozšiřováním ale stejně důležité je trénování útočných jednotek. Pokud se hráč zaměří na silnou ekonomiku a nebude mít dostatečné vojsko, může být překvapen nepřátelským vojskem v brzkých minutách a ihned poražen. Na druhou stranu, pokud se hráč zaměří na tréning vojska v co nejkratším čase, může vyhrát, pokud nebude jeho útok chytře odražen. Pokud se tak stane, bude poté velice pozadu co se týče ekonomiky a soupeř, pokud se zameřil na ekonomiku bude mít více surovin pro silnější a početnější armádu.

Q-learning potřebuje získat stavy a akce, které může vykonávat. V tomto případě lze ze hry vytvořit stavy pomocí aktuálně dostupných informací. Lze vybrat kombinaci několika hodnot, jako je například počet surovin, velikost vojska a populační limit. Ve *Starcraftu II* je okolo 800 akcí, které jsou dále rozděleny do jednotlivých ras. Každá akce má ještě své argumenty. Pokud vybereme například akci *Výběru* máme k dispozici 2 argumenty, kde první hodnota označuje, jaký typ výberu požadujeme. Můžeme vybírat jeden bod, skupinu stejných jednotek nebo výběr pomocí regionu. Pro Q-learning lze vybrat několik jednoduchých akcí a vybírat odměnu podle času, kdy byla akce vykonána. Tímto můžeme po agentovi požadovat, aby se snažil co nejrychleji postavit nějakou budovu, případně aby začal efektivně trénovat jednotky a přitom vytvářet další dělníky.

Rozhraní Starcraftu pro strojové učení obsahuje spoustu různých informací, které mají agentovo vidění co nejlépe přiblížit lidskému. Jedna z věcí, kterou má ovšem agent k dispozici je i současné skóre, které běžný hráč nevidí, dokud hra neskončí. Toto skóre slouží pro určování odměny za vybranou akci.

Kapitola 3

Implementace

3.1 Návrh

Prvotní návrh implementace byl takový, že agent bude mít k dispozici pár příkazů, pomocí kterých bude moci ovládat hru. Tuto sadu příkazu se bude snažit vykonávat tak, aby dosáhl největší odměny, pokud by byla odměna možná a zároveň aby dosáhl výhry. K dispozici měl pár informací, co se týče pozorování a byl testován na testovací mapě *Simple64* 3.5.

Tento agent měl plno nevýhod. Jednou z nich bylo, že nemohl pohybovat s kamerou, jelikož vše bylo soustředěné okolo hlavní budovy. Mým konečným návrhem bylo změnit agentovu rasu na *Zerg* a vytvořit několik Q-learning sítí, které budou navzájem spolupracovat.

3.2 S2Protocol

S2Protocol je protokol, který obsahuje balík knihoven a aplikací, které umožňují komunikaci s hrou *Starcraft II*. Tento protokol umožňuje dekódovat různé datové struktury a události, které hra odesílá pomocí svého vnitřního protokolu agentům. Tento protokol dekóduje:

- Hlavičku záznamů z her
- Detail o právě hrané hře
- Detail o právě přehrávaném záznamu
- Události ve hře

Tento protokol se využívá pro vytváření agentů, kteří se učí z pozorování. Z toho důvodu tento protokol nezahrnuje žádné informace týkající se protihráčů, ani žádné informace, které by neměl obyčený hráč. Tento protokol vyžaduje přesné specifikování právě přehrávaného záznamu či přehrávané hry z důvodu různých datových struktur pro různé verze hry.

3.3 PySC2

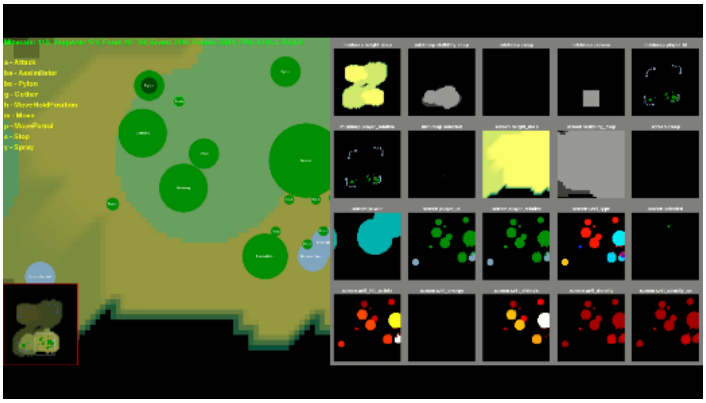
PySC2[1] je framework od společnosti *Google DeepMind*, který je postavený nad s2protocol od společnosti *Blizzard*. Slouží jako nádstavba pro vytváření agentů strojového učení.

[1] PySC2 repozitář: https://github.com/deepmind/pysc2

Jedna z nejdůležitějších částí, kterou poskytuje framework *PySC2* je pohled na hru pro agenta. Jelikož by bylo velice obtížné rozpoznávat úrovňové rozdíly, jednotky, barvu jednotek apd... poskytuje framework různé vrstvy, které představují pohled na hru pro strojové učení.

Na obrázku 3.1 je na levé polovině vykreslená hra pomocí těchto vrstev a na pravé polovině je většina vrstev, které znázorňují různé pohledy na mapu, jako například která část mapy je ještě neodhalena, která byla odhalena a kde je aktuální pohled. Dále obsahují pohledy kamery, jako rozlišení jednotlivých jednotek jako přátelských, nepřátelských, neutrálních a vlastních.

Všechny tyto pohledy představují uvnitř frameworku dvourozměrná pole, vyplněné hodnotami odpovídajícím například identifikátorům jednotlivých jednotek, případně jednociferná čísla pro rozlišení frakce jednotek.



Obrázek 3.1: Poskytované vrstvy pro agenta

PySC2 obsahuje plno nástrojů a knihoven, které předvádějí použití tohoto frameworku. Mezi základní spustitelné nástroje patří:

- Agent
- Play
- Replay Actions
- Replay Info
- Play vs Agent

3.3.1 Agent

Nástroj *Agent* slouží ke spuštění agenta. Tento nástroj načte třídu obsahující kód a instrukce pro agenta, které začne vykonávat. Tento nástroj má spoustu přepínačů, které specifikují, jakou mapu, rasu, či verzi hry má nástroj načíst. Vytvořená třída s agentem funguje na principu jednoduché dědičnosti, kde agent musí obsahovat třídu, která dědí ze základní třídy *BaseAgent*. Tato třída poté musí přetížit metodu step, která se volá pro každou akci ve hře a musí vracet požadovanou akci. Pomocí přepínače *–agent* se specifikuje název souboru a třída agenta, který se má spustit. Dalším užitečným přepínačem je *difficulty*,

který určuje proti jaké obtížnosti předprogramovaného bota bude agent hrát. Ve hře je celkem 9 obtížností bota.

- Very Easy
- Easy
- Medium
- Hard
- Harder
- Very Hard
- Cheat Vision
- Cheat Money
- Cheat Insane

Ze všech těchto obtížností lze vybat úroveň oponenta, ale v normální hře lze hrát pouze proti prvním 6 obtížnostem.

3.3.2 Play

Nástroj *Play* má dvě možnosti spuštění. První možností, je spustit normální hru na zvolené mapě, kde framework vytvoří spojení s hrou, vykreslí všechny vrstvy, které jsou poskytovány pro agenta a zobrazí samotnou hru. Toto spuštění slouží pro testování a předvedení frameworku, kde hráč sám může vykonávat akce a ovládat spuštěnou hru.

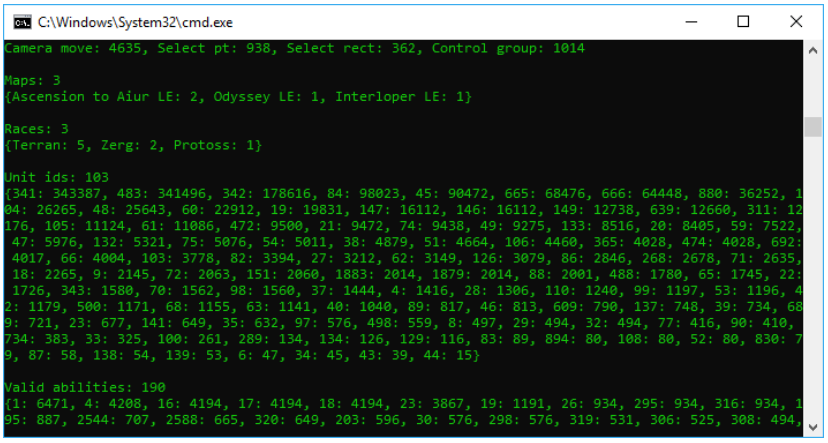
Druhou možností, je přehrání záznamu, kde se frameworku zadá nějaký existující záznam a on jej přehraje. Přehrávání záznamu funguje podobně, jako obyčejné spuštění a na první pohled vypadá uplně stejne. Jediný rozdíl je v tom, že se hra nedá ovládat a akce se vykonávají přesně podle instrukcí v přehrávaném záznamu z hry.

3.3.3 Replay Actions

Tento nástroj slouží k přehrávání již odehraných záznamů a sbírání informací. Nástroji se zadá cesta k jednomu nebo celé složce záznamů a poté se spustí hra *Starcraft II*, ve které se záznamy začnou přehrávat a zároveň do konzole se začnou vypisovat statistiky o tom, jaké všechny akce byly ve hře provedeny. Tento nástroj slouží jako příklad pro vytahování akcí ze záznamu a použití těchto akcích například v *Deep-Q-Network*.

Výhodou tohoto nástroje je, že dokáže pracovat paralelně. Pokud se zadá celá složka se záznamy, nástroj spustí hned několik instancí s hrou a začne všechny zpracovávat zároveň. Nástroj poukazuje také na rozdělení jednotlivých akcí do skupin, jako jsou použité ability, validní akce a také všechny identifikátory jednotek, které byly v záznamu použity.

Výstup tohoto nástroje je poněkud komplikovaný jak je možné vidět na Obrázku 3.3, ale co se týče zdrojového kódu, je jako demonstrace velice dobře čitelný a použitelný.

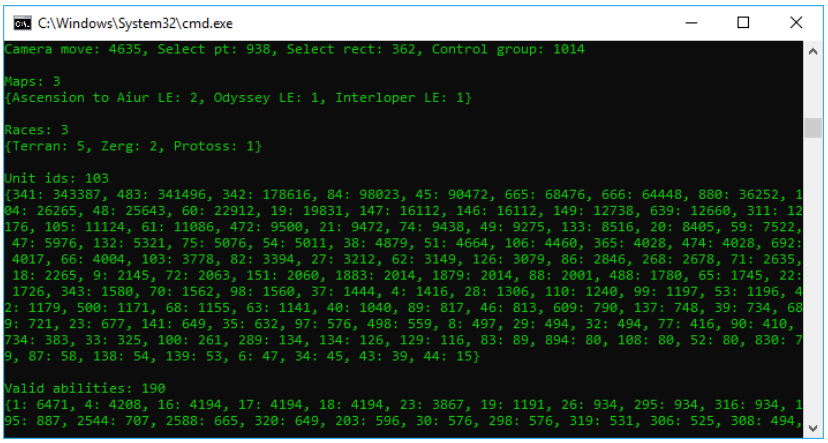


Obrázek 3.2: Výstup nástroje Replay Actions.

3.3.4 Replay Info

Nástroj *Replay Info* je velice podobný nástroji *Replay Actions* 3.3.3. Taktéž požaduje jako vstup jeden nebo více záznamů a taktéž spouší hru, ale důležitým rozdílem je, že tento nástroj nepřehrává celý záznam ze hry, ale pouze jej načte aby bylo možné ze záznamu vytáhnout informace jako kdo byl vítěz, jaká byla úroveň hráčů, kteří hráli a jaké je jejich hodnocení v oficiálních hodnocených hrách.

Tento nástroj slouží hlavně pro filtrování velké skupiny záznamů, podle zadaných parametrů. Stejně jako *Replay Actions* dokáže pracovat paralelně. Spustí několik instancí hry a do každé načte záznam, z kterého vytahuje informace.



Obrázek 3.3: Výstup nástroje Replay Actions.

3.3.5 Play vs Agent

Momentálně posledním nástrojem frameworku je *Play vs Agent*, což je nástroj, který umožňuje zahrát si proti nějakému naprogramovanému agentovi. Tento nástroj jsem sice testoval,

ale jelikož je celý framework stále ve vývoji, není tento nástroj zcela funkční a proto není ještě možné hrát.

3.4 Můj první agent

Úplně první agent byl označen názvem *MyBot*. Hrál za herní rasu Terran a měl k dispozici stavění budovy *Supply Depot*[2] a *Barrack*[3]. Dále měl k dispozici trénování dělníků a základní útočné jednotky *Marrine*[4].

Pozorování bylo definováno hodnotami, jako jsou počet dělníků, počet útočných jednotek, počet natěžených surovin, počet postavených budov a dostával odměnu za zabití nepřátelkých jednotek, za vytvoření budovy a vytváření dělníků podle populačního limitu.

Cílem tohoto agenta bylo co nejoptimálněji vybírat akce, které se mají provádět, aby bylo dosaženo co největšiho počtu útočných jednotek v co nejkratší dobu. Tato implementace dovolovala maximální populační limit 22 jednotek, jelikož pozice staveb byly dány staticky a jakmile byla jednou budova postavena, nemohla se postavit další. Tento problém jsem později vyřešil nádhodným vybíráním souřadnice na obrazovce, čímž jsem docílil teoreticky neomezeného stavění požadovaných budov. Problém tohoto návrhu bylo, že při "správném" postavení budov se zamezilo odchodu útočných jednotek a byly doslova uvězněny na svém vlastním území.

3.5 Prvotní testy

Prvotní testy probíhaly na mapě s názvem *Simple64*, kterou je možné vidět na Obrázku 3.4. Mapa je rozvržením klasická herní mapa ale bez jakýchkoli ozdobných doplňků. Obsahuje pouze dvě místa, kde lze rozšířit ekonomiku. Mapa je souřadnicemi přesný čtverec o velikosti jedné délky 64.

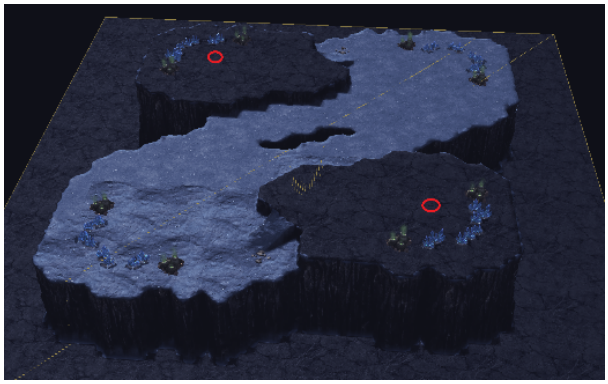
Tato mapa je využívána pro obecné testování agentů. Prostředí je jednoduché a souřadnice jsou přesně dané. Proto není potřeba žádných složitých výpočtů, pro určení pozice a případného pohybu s kamerou. Mapa obsahuje pouze jednoduchý terén, který je pro agenta lehce detekovatelný.

Na této mapě bylo testováno odesílání jednotek na určené souřadnice, podle polohy agentovy základy. Jelikož mapa diagonálně symetrická, stačilo zkontrolovat souřadnice a případně je převrátit po diagonále.

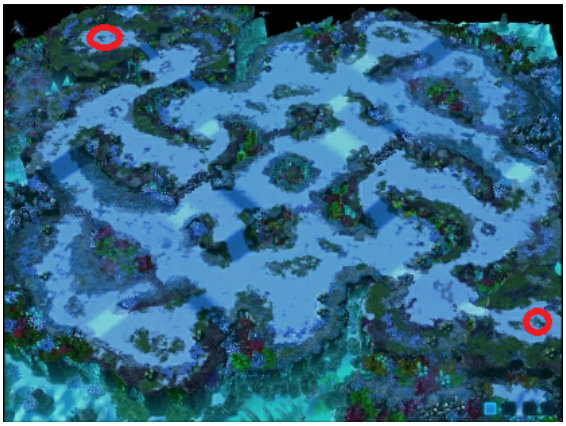
3.6 AbyssalReef

Cílem bylo implementovat agenta tak, aby dokázal odehrát zápas na mapě *Abyssal Reef*, která se hraje i na hodnocených zápasech. Tato mapa je velice komplikovaná a obsahuje mnoho různých překážek. Je mnohem větši, než mapa *Simple64* 3.5, má velice komplikovaný terén a obsahuje několik výškových stupňu.

[2]Budova pro zvyšování populačního limitu
[3]Základní budova pro trénování jednotek.
[4]Základní střelecká jednotka od rasy Terran



Obrázek 3.4: Mapa Simple64



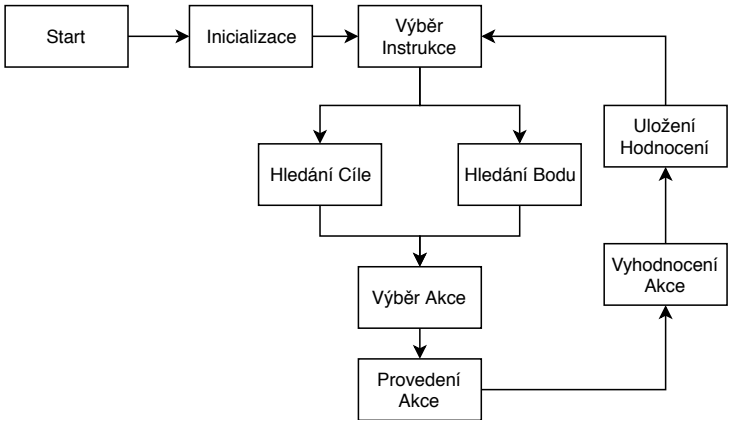
Obrázek 3.5: Mapa Abyssal Reef

3.7 QAgent

Implementace mého agenta obsahuje 4 Q-learning algoritmy. Jeden algoritmus se stará o kameru 3.7.1, druhý se stará o hledání jednotek na obrazovce 3.7.2, třetí se stará o výběr pozice pro útok 3.7.4 a poslední vybírá akce, které se mají provést.

Agent nejprve provede inicializaci. Resetuje všechny proměnné ovlivňující prostředí, najde hlavní budovu a postaví budovu pro trénovaní základní útočné jednotky. Další proces je výběr akcí, kde začíná agentovo učení. Agent zvolí nějakou instrukci, která se provede a podle výsledku se ohodnotí. Pokud je zvolena instrukce, která požáduje nalezení nějaké budovy, přebere kontrolu síť s vyhledáváním pomocí kamery, kde se vybere bod, kde by se budova mohla nacházet a spustí se akce přesunu kamery. Pokud na zvolených souřadnicích budova je, agent dostane kladné hodnocení, které je poté uloženo. Pokud budova nebyla nalezena, hledání pokračuje.

Agent pro výber akce má k dispozici sadu instrukcí, které jsou v tomto případě interpretovány jako akce. Tyto instrukce obsahují sadu příkazu, které se mají provést. Jako stav pro každou akci je kombinace hned několika pozorování ze hry jako je maximální populační limit, kolik populace je použito, kolik zabírá armáda a kolik zabírají dělníci. Když



Obrázek 3.6: Schéma Agenta

agent vybere například akci pro vytvoření základní útočné jednotky, provedou se následující instrukce:

- Vybrat hlavní budovu
- Vybrat larvy[5]
- Trénovat jednotku Zergling[6]

Jakmile se dokončí tato sada příkazů, agent vybírá další akci. Všechny tyto Q-learning algoritmy jsou propojeny. Pokud se vybere tato akce pro tréning jednotek, převezme kontrolu ovládání kamery, které má zaúkol najít hlavní budovu. Poté, co je hlavní budova nalezena, je na řadě hledání pozice na obrazovce, která obdrží pozici hlavní budovy a kamery. Poté, co se najde hlavní budova a vybere se, začne tréning jednotek. Pokud se vybere akce pro útok musí se nejprve vybrat celá armáda, poté se hledá pozice pro útok a nakonec se odešle celá armáda na toto vybrané místo.

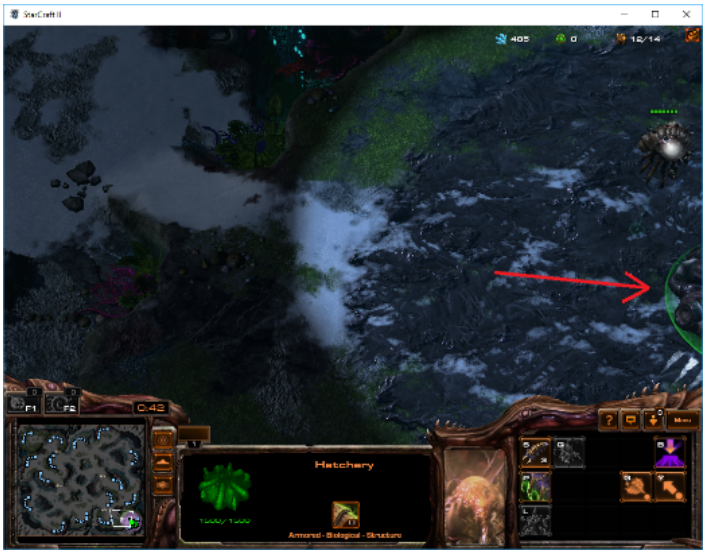
3.7.1 Ovládání kamery

Implementace kamery byla původně jedna z nejkomplikovanějších věcí. Pro ovládání čehokoli ve hře je potřeba, aby to bylo vidět na obrazovce, neboli aby to viděla kamera.

Původní návrh agenta, který měl k dispozici ovládání kamery a mohl libovolně měnit jeji pozici, nedokázal najít potřebné body pro operaci a agent dělal jen to, že stále měnil pozici kamery a nemohl nic dělat. Toto bylo velice neefektivní, jelikož je obtížné bez nějaké zpětné vazby nasměrovat agenta, aby věděl, že má kameru na správném místě.

Druhý návrh ovládání kamery spočíval v tom, že agent zadal instrukci, kterou chtěl provést, například vybrat nějako budovu a tato instrukce v sobě obnášela i hledání požadované budovy. Prvním krokem bylo najít tuto budovu a zobrazit jí na kameře. Zde agent věděl, co má hledat a měl k dispozici všechny dostupné pozice pro přesun kamery. Pokud byla vyhledávaná věc na obrazovce, agent dostal odměnu za nalezení budovy, čímž si zapamatoval, že zde se budova nachází.

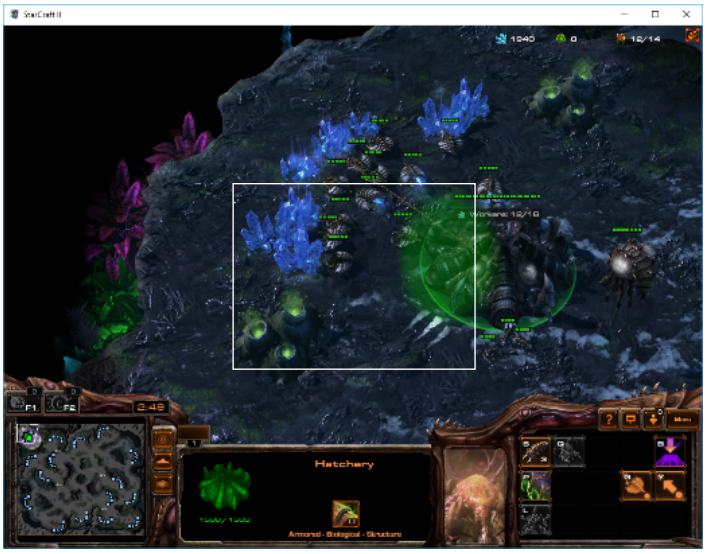
[5]Jednotka, pomocí které se vytváří další jednotky pro rasu Zerg.
[6]Základní útočná jednotka pro rasu Zerg.



Obrázek 3.7: Špatný pohled na hlavní budovu

Ovšem problém tohoto návrhu byl v tom, že agent sice svůj cíl nalezl, ale často se stávalo, že tento cíl byl někde na kraji obrazovky, takže jej skoro nebylo možné vidět. Jak je vidět na Obrázku 3.7. Agent požadovaný cíl, neboli hlavní budovu úspěšně našel, ale je na velmi malé části obrazovky, což je pro *Hledání jednotek* 3.7.2 na obrazovce komplikované.

Tento problém vyřešil další návrh, kde agentovi byla ořezána kamera. To mělo zapříčinit lepší nastavení kamery, což donutilo agenta aby se snažil hledaný cíl vycentrovat na kameru. To již bylo přijatelné, jelikož každé hledání nějaké určité jednotky skončilo tak, že hledaná jednotka byla někde okolo středu obrazovky.



Obrázek 3.8: Pohled na hlavní budovu s omezeným pohledem

Jak je vidět na obrázku 3.8, agent má pro hledání věcí na mapě omezenou kameru. Touto restrikcí je docíleno lepšího pohledu na hledaný cíl. Ovšem stále to není to nejlepší, a proto byla vytvořená další verze, která se snaží hledaný cíl umístit co nejlépe do středu obrazovky.

Nejnovější implementace vyhledávání je inspirována algoritmem hledáním jednotek na obrazovce 3.7.2. Princip vyhledávání spočívá v tom, že po nalezení požadovaného cíle se spočítá pokrytí ve vyhledávací mřížce 3.7.5 a toto pokrytí se poté použije jako odměna pro vyhledávací algoritmus.

Obrázek 3.9: Pohled na hlavní budovu, která je lépe vycentrovaná.

Čím lépe vyhledávání vycentruje kameru na požadovaný cíl, tím větší odměnu dostane. Tímto principem se vyhledávání snaží co nejlépe vycentrovat kameru na hledaný cíl a tudíš nekončí pouze u nalezení, ale také se snaží kameru upravit.

Jak je možné vidět na obrázku 3.9, agent se snaží co nejlépe upravit kameru tak, aby vzhledávaný cíl pokryl co největší část mřížky. Tím docílí toho, že dostane nejvyšší odměnu a zároveň se snaží uchovat nejpřesnější pohled.

Jeden z klíčových problémů, s kterými jsem se setkal, byla absence rozlišování vlastních a nepřátelských budov. Pokud protivník hrál za stejsou rasu, jako agent, tedy *Zerg* a na mapě byla nalezena jeho hlavní budova, mohl nastal problém, kdy agent vyhledával tuto budovu místo své vlastní. Tímto se agent naučil hledat hlavní budovu na místě, kde nebyla. Po opravení tohoto problému a opětovné spuštění s již naučeným špatným místem, se agent okamžitě začal přeučovat a v podstatě zapomínat toto špatné místo.

3.7.2 Hledání jednotek

Pro vybrání nějakého cíle, například hlavní budovy, je zapotřebí znát pozici tohoto cíle na obrazovce. *PySC2* 3.3 poskytuje několik dvourozměrných polí, které znázorňují pohled na obrazovku. Tento pohled neposkytuje souřadnice, ale slouží pro kontrolu, co je na zadaných souřadnicích.

Hledání jednotek je druhým krokem pro jakoukoli instrukci nalezení něčeho na obrazovce. Tento algoritmus je závislý na ovládání kamery 3.7.1. Pokud agent vybere instrukci

pro nalezení nějaké požadované jednotky nebo budovy, je nejdříve spuštěno ovládání kamery, které se pokusí nalézt cíl nejprve na obrazovce a poté předá souřadnice kamery algoritmu pro hledání jednotek, který má k dispozici mřížku 84x84. Na této mřížce se snaží najít body, které po kontrole v poskytnutém pohledu odpovídají požadovanému cíli.

Prvotní implemetnace této vyhledávající sítě spočívala v tom, že síť dostala od ovládání kamery 3.7.1 souřadnice, které použila pro vytvoření stavu a poté hledala na obrazovce požadovaný cíl. Tato síť nepotřebovala žádnou zvláštní interakci s hrou, jelikož zvolené body mohla hned testovat, zda se trefila. Tato implementace byla funkční ale nebyla vždy přesná. Pokud se stalo, že například hlavní budova byla zakryta nějakou létající jednotkou, mohlo se stát, že zvolený bod již nebyl validní a síť musela hledat další.

Nejnovější implementace hledání jednotek pracuje podobně. Vybere bod na obrazovce, ale pro testování, zda se síť trefila do požadovaného cíle se vytvoří vyhledávací mřížka 3.7.5 o velikosti 3x3, kde střed mřížky jsou vybrané souřadnice sítí. Pomocí této mřížky se spočítá, jaké je pokrytí cíle a tato hodnota se použije jako odměna pro síť. Touto implementací je docíleno přesnějšího vybírání, jelikož se síť snaží vybírat místo, které je spíše středem hledaného cíle, než například jeho okraj.

3.7.3 Speciální případ hledání

Při hledání hlavni budovy nastával problém, kde při vytvoření velkého množství *Overlordů*[7] nebylo velmi možné vybrat hlavní budovu a algoritmus pro hledání jednotek na obrazovce ji nemohl najít. Tento problém byl vyřešen použitím skupin ve hře.

Skupiny jsou při hraní nastaveny pomocí vybrané klávesové zkratky a jejich účel je označení jednotek, případně přesunutí kamery k označenému cíli. Tuto funkci jsem použil pro výber hlavní budovy, čímž jsem uvolnil kameru od statického sledování jednoho místa ve hře a využil jsem této volnosti k zobrazování míst k útoku. Pokud algoritmus pro hledání bodu k útoku 3.7.4 vybere místo, kde se má zaútočit, je zde přesunuta kamera a lze sledovat přesun jednotek a případný útok.

3.7.4 Hledaní bodu pro útok

Implementaci útoku lze vyřešit několika způsoby. Jelikož jsou mapy většinou diagonálně převrácené, stačí zjistit, na jaké straně hraje agent a poté poslat útok na místo převrácené o tuto diagonálu.

Tento způsob útoku má však jednu obrovskou nevýhodu. Pokud nepřítel postaví budovy, které nejsou v cestě útoku, nebo pokud si postaví základní budovu na nějaké nové místo, není možné ji zaměřit a tedy eliminovat nepřítele.

Jak je možné vidět na obrázku 3.10, statický bod útoku nedokáže eliminovat nepřítele. Tímto dostává nepřítel příležitost k obnově svých budov, postavení nové armády a novou šanci na výhru.

Tento problém jsem vyřešil implementací další sítě, která se stará o hledání bodu pro útok. Princip je velmi podobný jako hledání jednotek 3.7.2, s tím rozdílem, že se nepracuje s obrazovkou, ale minimapou.

Při každém hledání bodu pro útok se vytvoří stav, kde se postupně kontroluje mapa rozdělená na 16x16 polí a ukládají se hodnoty 0 a 1 podle toho, zda se na konkréním místě nachází, nebo byla objevena nějaká nepřátelská budova. Po uložení tohoto stavu agent vybírá místo k útoku, kde má k opět dispozici souřadnice 16x16. Jakmile agent vybere

[7] Jednotka, která zvyšuje populační limit

Obrázek 3.10: Nedosažitelná nepřátelská budova.

vhodné místo, spočítá se pokrytí nepřítelem na daném místě a tento výsledek se uloží odměna pro agenta.

Jak je možné vidět na obrázku 3.7.4, pomocí této implementace pro útok je pro agenta velice jednoduché najít to správné místo a eliminovat nepřítele.

3.7.5 Vyhledávací mřížka

Implementace *vyhledávací mřížky* dopomohla jednak k přesnějšímu zobrazování hledaných cílů a taktéž ke generování hodnocení. Každý algoritmus, který používá vyhledávací mřížku, upravuje vrstvu poskytovanou frameworkem *PySC2*, kde vezme vybranou vrstvu vyobrazenou jako dvourozměrné pole, vybere bod v tomto poli a vytvoří z něj a jeho okolí mřížku.

$$\begin{bmatrix} 4 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{7} & \mathbf{7} & 0 & 0 & 3 & 0 \\ 0 & \mathbf{7} & \mathbf{7} & \mathbf{7} & \mathbf{7} & 0 & 3 & 0 \\ 0 & \mathbf{7} & \mathbf{7} & \mathbf{7} & \mathbf{7} & 0 & 0 & 3 \\ 0 & 0 & \mathbf{7} & \mathbf{7} & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 2 & 0 & 2 & 0 & 3 & 0 & 0 \end{bmatrix}$$

Obrázek 3.11: Ukázka vrstvy s jednotkami

Vrstva zobrazující identifikázory jednotek může vypadat jako na obrázku 3.11. Řekněme, že číslo 7 představuje hlavní budovu, číslo 2 představuje základní stavební jednotku, v případě mého agenta je to *Drone*[8], číslo 3 představuje *minerály*[9] a číslo 4 představuje *Spawning Pool*[10].

[8]Drone je základní stavební jednotka z rasy Zerg
[9]Minerály jsou základní surovina používaná ve hře pro stavbu
[10]Budova rasy Zerg, která zpřístupňuje trénování základní útočné jednotky Zergling

Na tuto matici je použit skript, který převede všechny vyhledávané hodnoty na 1 a všechny ostatní hodnoty na 0. Tím se vytvoří nová matice 3.12.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Obrázek 3.12: Aplikovaný filtr na jednu jednotku (Hlavní budova)

Z této matice se již jednoduše vypočítá pokrytí, kde se sečtou všechny hodnoty obsažené v matici a podělí se velikostí matice. Tímto principem se odměňuje například umístění kamery na mapě agenta, pokud hledá hlavní budovu. V tomto případě je matice velká 8x8 a hlavní budova může maximálně obsadit 12 míst a to znamená, že agent může jako největší odměnu získat 0.1875 ale pokud by se stalo, že agent vybere pro kameru místo, kde hlavní budova bude zabírat pouze jedno místo, jeho odměna bude 0.015625.

Jelikož se agent snaží vždy vytěžit z vybrané akce co nejvíce, bude stále upravovat kameru dokud nedosáhne co největší možné hodnoty. V tomto případě to bude vždy zmiňovaných 0.1875 a takto se naučí, kde má kameru umístit.

3.8 Jiný agent

Můj agent je implementovaný pomocí 4 Q-learning algoritmů, které spolu spolupracují. Touto implementací jsem chtěl dosáhnout spolupráce těchto algoritmů a ty, které zrovna nejsou potřeba aby nebránili v práci ostatním. Hlavní algoritmus je vybírání akcí, který oslovuje ostatní sítě a dává jim za úkol podakci, které je potřebná k provedení instrukce.

3.8.1 Jiný agent 1 - chris-chris

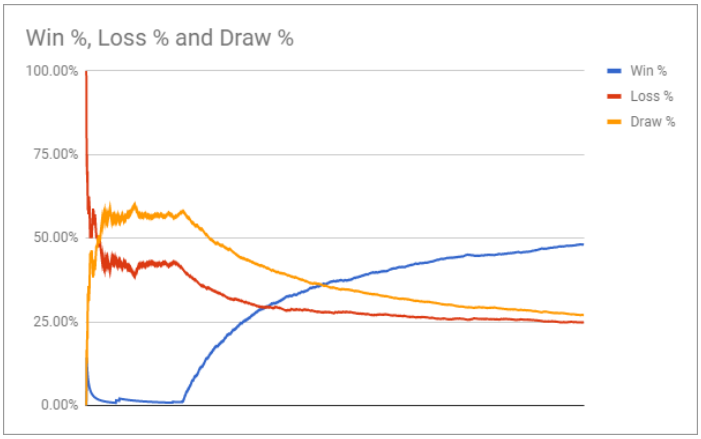
První implementací, kterou jsem si vybral k porovnání s mým agentam, je od *Chris Hoyean Song*[11] (přezdívkou chris-chris), který implementoval agenty pro minihry, které jsou poskytovány frameworkem *PySC2*. Tito agenti jsou specializováni na ovládání jednotek a určování místa přesunu na jedné obrazovce. Tím je myšleno, že agent sleduje pomocí kamery stále jedno místo a učí se co nejefektivněji splnit úkol, který na dané minihře je.

3.8.2 Jiný agent 2 - skjb

Druhou implementací, kterou jsem vybral k porovnání, je od *Steven Brown*[12] (přezdívkou skjb), který implementoval agenta pomocí Q-learning, pro hraní proti předprogramovanému botovi. Tato implementace je hodně blízká mé, kde je pomocí Q-learning vybíráno mezi akcemi, které jsou k dispozici a tyto akce jsou následně prováděny.

[11] Github link na profil Chris Hoyean Song: https://github.com/chris-chris
[12] Github link na profil Steven Brown: https://github.com/skjb

Agent hraje za rasu *Terran* a hraje proti úrovni *Very Easy*. Úspěšnost tohoto agenta je znázorněna na grafu



Obrázek 3.13: Míra vítězství agenta od Steven Brown
Zdroj:https://itnext.io/refine-your-sparse-pysc2-agent-a3feb189bc68

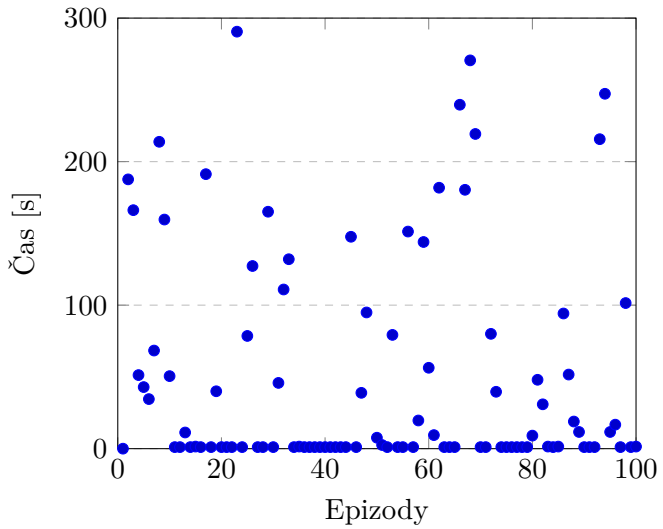
Kapitola 4

Experimentální vyhodnocení a diskuse

V experimentech, které jsem prováděl s agentem, jsem se zaměřil na implementaci sítí, které bylo možné testovat s každou hrou. Síť, kterou jsem ale netestoval, byla síť pro hledání jednotek na obrazovce 3.7.2, jelikož tato síť nepotřebuje ve hře odesílat žádné akce, ale pouze získá vrstvu poskytovanou frameworkem *PySC2* a na této vrstvě pracuje. Ostatní sítě při výberu akce odesílají pomocí protokolu instrukci do hry, která se provede. Výsledky z experimentů jsou seskupovány do epizod. Jedna epizoda znamená jedna odehraná hra. Čas v experimentech je herní čas, který je 8x zrychlený oproti reálnému času.

4.1 Experiment 1 - hledání hlavní budovy

Prvním experimentem, bylo měření rychlosti sítě pro vyhledávání jednotek na mapě, neboli ovládání kamery 3.7.1. Tato síť je velice důležitá pro agenta, jelikož úplně prvním krokem je nutnost nalezení hlavní budovy. Při úspěchu je již hlavní budova uložena do skupiny, odkud se poté vybírá.

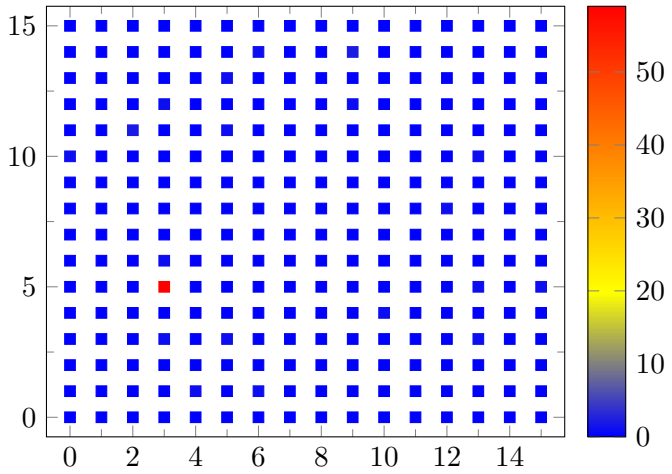


Obrázek 4.1: Čas potřebný k nalezení budovy

Jak je možné vidět na grafu 4.1, hledání hlavní budovy se ve většině případů blížilo k 0. V úplně první epizodě agent nedokázal včas najít hlavní budovu a byl poražen, v každé další epizodě již byl úspěšnější.

4.2 Experiment 2 - Vybírání pozice pro útok

Druhý experiment je soustředěn na vybírání cíle pro útok. Jelikož agent od první epizody neví, kde nepřítel je, musí ho sám najít. K tomu slouží síť pro hledání bodu útoku 3.7.4, která musí nejprve hádat, kde nepřítel je a pokud jej nalezne, poznamená si tuto pozici a při výběru dalšího bodu bude spíše vybírat tento úspěšný bod. Data pro tento experiment jsou rozdělena do 3 částí. První část je vybírání bodu útoku v první epizodě, tzn. že agent vůbec neví, kde se nepřítel nachází. Druhá část experimentu je v vybírání bodu po 50 epizodách, zde už je agent rozhodný a vybírá místa, o kterých ví, že se zde nepřítel nachází. Třetí část experimentu je výber bodu po 1000 epizodách, kde už si je agent jist a útočí s jistotou na vybraný bod.

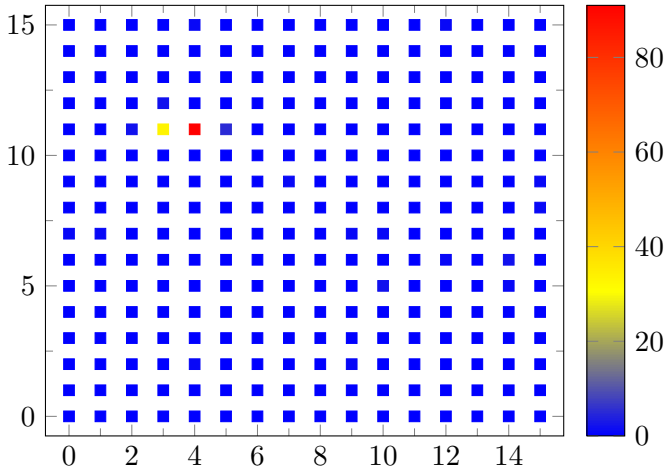


Obrázek 4.2: Vybírání místa pro útok v první epizodě.

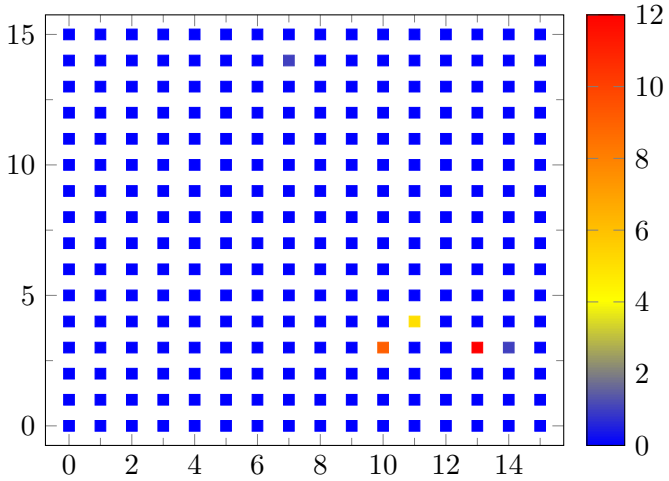
Na Obrázku 4.2, který znázorňuje vybírání místa v úplně první epizodě, kde agent neměl k dispozici žádné předešlé zkušenosti, agent často vybíral jedno specifické místo. Toto místo má souřadnice $(3, 5)$ a bylo vybráno celkem 59 krát. Toto místo je zvláštní tím, že není důvod, proč by na toto místo agent útočil, jelikož se zde nenachází žádná nepřátelská budova ani jednotka. Agent ale dostává odměnu, pokud vybere místo, kde se nějaká nepřátelská jednotka nachází. Vysvětlením pro toto chování je, že agent zrovna hrál proti stejné rase, jako je agent sám, čili *Zerg* a nepřátelský bot poslal jednotku *Overlord* na průzkum. Agent tuto jednotku našel na tomto místě a dostával za to odměnu, jelikož na ni chtěl zaútočit. Tato jednotka není pozemní, ale letecká, tzn. že agent opětovně útočil na toto místo, jelikož nebych schopen tuto jednotku eliminovat.

Na Obrázku 4.3, který znázorňuje vybírání místa v 50. epizodě je vidět, že agent již přesně útočí na místo, kde se nachází nepřátelská základna. Našel si dvě místa na mapě, které považuje za klíčové body pro útok a na tyto body opětovně útočí.

Na obrázku 4.4, který znázorňuje vybrané místa v 100. neboli poslední epizodě je vidět, že agent útočil nejen na místo, kde se nachází nepřátelská základna, ale také na místo, kde



Obrázek 4.3: Vybírání místa pro útok v 50. epizodě.



Obrázek 4.4: Vybírání místa pro útok v 100. epizodě.

nepřítel postavil rozšiřující základnu a budovu za ní. Proto je na obrázku znázorněn jeden bod v pravém dolním rohu, který představuje hlavní nepřátelskou základnu a poté dva další body, které jsou blíže středu, představující rozšíření.

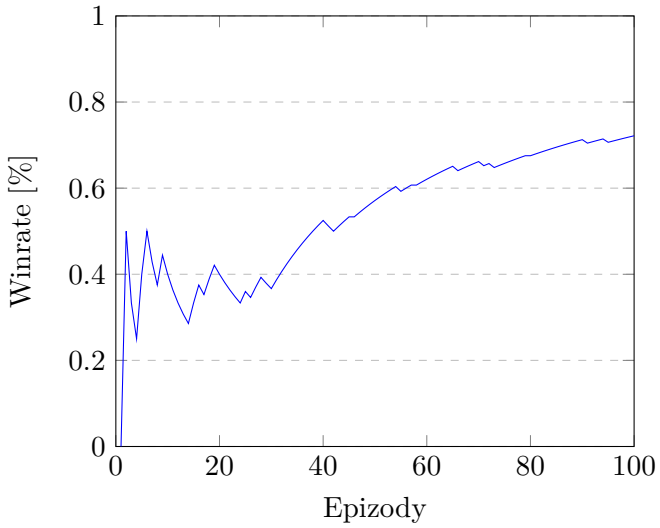
4.3 Experiment 3 - Vybírání akcí

Třetí experiment s názvem vybírání akcí je zaměřen na úspěšnost sítě, která vybírá akce, které se mají ve hře provádět. Úspěšnost této sítě jsem se rozhodl měřit v poměru výhry ku epizodám, neboli zahraným hrám. Tento experiment je rozdělen do 5 částí. V každé části je agent připraven o vše, co se naučil a začíná se znova učit.

U této sítě jako jediné jsem se rozhodl experimentovat s různými úrovněmi nepřítele, jelikož u této sítě na tom záleží. U ostatních sítí není důležitá úroveň nepřítele, ale přesnost a rychlost zpracování.

4.3.1 Velmi jednoduchý protivník

V první části agent soupeří proti předprogramovanému botovi ve hře s obtížností *Very Easy*[1]. Tento bot je velmi pomalý, nezaútoči před 10. minutou a nestaví žádné pokročilé technologie.



Obrázek 4.5: Very Easy

Jak je vidět na obrázku 4.5 při hraní proti úrovni *Very Easy* si z počátku agent držel 40% vítězství. Po asi 30 epizodách se již agent naučil rychlé stavění a trénovaní jednotek a jeho mírá vítězství až na nějaké vyjímky rostla. V poslední 100. epizodě měl nejvyšší míru vítězství a to 72.16%.

4.3.2 Jednoduchý protivníka

V druhé části agent soupeří proti těžšímu botovi, který je ve hře označován úrovní *Easy*. Tento bot je již rychlejší, co se týče tréningu jednotek a útočí mnohem dříve.

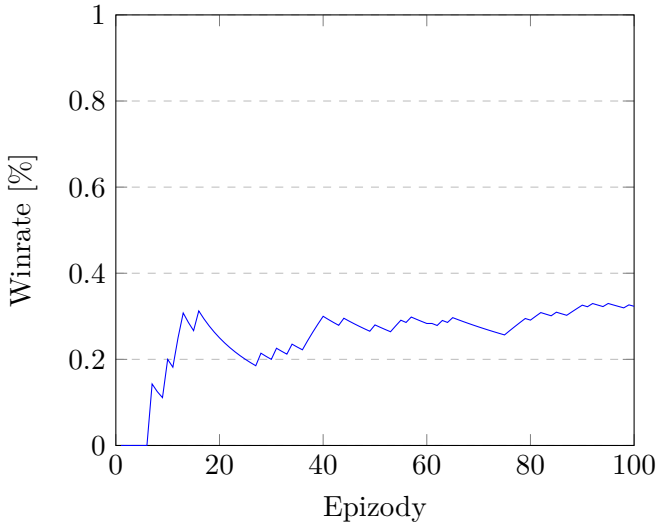
Na obrázku 4.6 je graf míry vítězství proti úrovni *Easy*. Při hraní proti úrovni s označením *Easy* agent poprvé vyhrál až v 7. epizodě. Agent se stále držel okolo 30% míry vítězství ale nejvíše se dostal v 95. epizodě, kde dosáhl 32.98%.

4.3.3 Střední úroveň protivníka

V třetí části agent soupeří proti mnohem těžšímu botovi, který je ve hře označován úrovní *Medium*. Tento bot je již rychlejší, co se týče tréningu jednotek a staví pokročilejší budovy.

Na obrázku 4.7 je znázornění epizod proti úroveni *Medium*. Zde již agent nebyl natolik úspěšný a jeho první výhra se uskutečnila až v 40. epizodě. Tento soupeř byl pro agenta již jakousi hranicí, kde již bylo poznat, že pouze základní útočné jednotky nejsou dostatečné pro udržení se ve hře, a pokud agent nezaútočil ve chvíly, kdy nepřítel nebyl ještě připraven, nebo neměl postavenou nějakou základní obranu, neměl šanci hru vyhrát.

[1]Very Easy je nejjednodušší možnou úrovní protivníka.



Obrázek 4.6: Easy

4.3.4 Těžká úroveň protivníka

Ve čtvrté části experimentu agent soupeří proti těžkému botovi, který je označován *Hard*. Tento bot je velmi rychlý, co se týče tréningu jednotek, ale také dělá všechna vylepšení a staví všechny budovy, které jsou ve hře k dispozici.

Na obrázku 4.8 je znázorněna míra vítězství proti úrovni *Hard*. Od tohoto experimentu se původně vítězství neočekávalo, jelikož úroveň hard je již úrovní, která dokáže odrazit většinu prvotních útoků a tím nedává agentovi moc možností, jak vyhrát. Překvapením bylo, když agent poprvé dokázal porazit tohoto nepřítele v 70. epizodě, kde se agentovi povedlo perfektně provést všechny potřebné akce včas a zničit ekonomiku nepřítele tak, že se již nemohl vrátit do hry. Nejvyšší míru vítězství, která agent dosáhl bylo 3,52% v 85. epizodě.

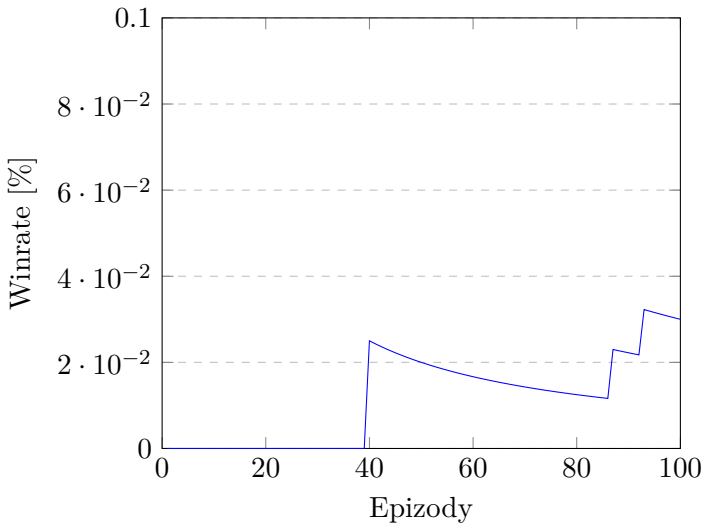
4.3.5 Nejtěžší úroveň protivníka

V poslední části experimentu agent soupeří proti nejtěžtímu botovi, který je označován *Hardest*. Tento bot je velmi rychlý, co se týče tréningu jednotek, ale také již dělá všechna vylepšení a staví všechny budovy, které jsou ve hře k dispozici.

Agent měl proti *Hardest* (Obrázek 4.9), neboli nejtěžší úrovni bota, podobné výsledky jako proti úrovni *Hard*. Jelikož tato úroveň již staví vše s předepsanými správnými časy, nebylo očekáváno, že by tentokrát agent dokázal vyhrát. Ještě větším překvapením, než proti úrovni *Hard* bylo, že agent dokázal načasovat a odeslat útok v momentě, kdy to ani tento bot nedokázal odrazit a to již v 56. epizodě. V tomto zápasu se agent lépe naučil časování a útočení, čímž dosáhl až 4.05% míry vítězství.

Výsledeky experimentů nekorespondovaly s očekáváním. Prvotní pokusy s agentem vyžadovaly mnohem více času, než agent začal být alespoň z části úspěšný. I přesto nedosahoval výsledků, jakých dosahuje nynější implementace.

Mým hlavním cílem byl agent, který dokáže jednou za čas vyhrát a zároveň vykazovat známky učení. Toho nebylo velmi dosaženo v první implementaci i přesto, že agent měl k dispozici jen pár akcí. I po několika epizodách byly vybrané akce stále chaotické a



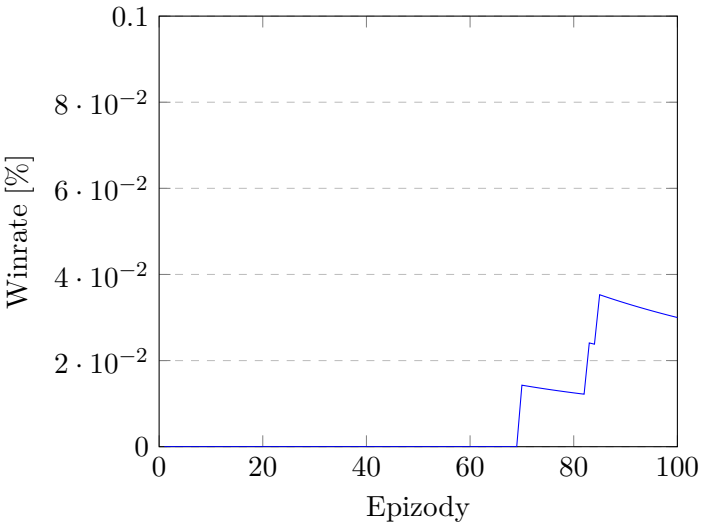
Obrázek 4.7: Medium

nepromyšlené. Agent stále dělal dojem náhodného vybírání, místo nějakých promyšlených tahů.

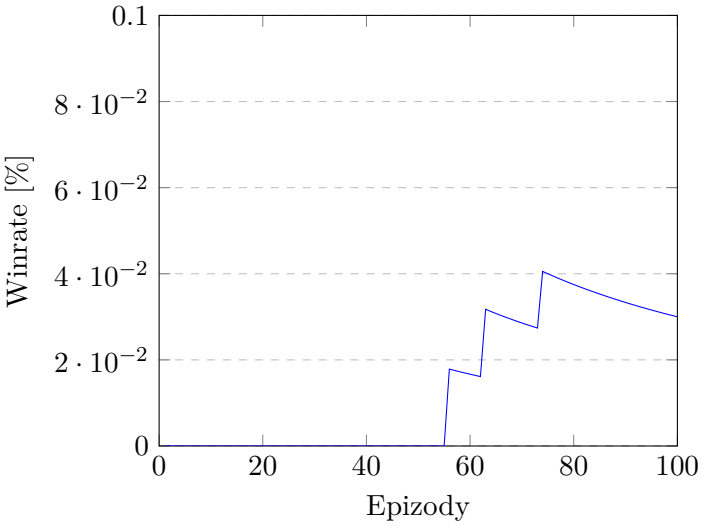
Nejnovější agent se od tohoto velice liší. Z počátku je znát, že zkouší náhodné akce a hledá řešení, za které bude nějak ohodnocen. Po několika epizodách je znát, že agent našel posloupnost instrukcí, které jsou podle něj optimální a dává to znát při jejich výběru.

Z počátku trénuje *Overlordy*[2], jelikož je to akce, která je možná a není za ni agent potrestán. Po několika málo epizodách se začíná zaměřovat více na trénování útočných jednotek a co nejrychlejší eliminaci nepřítele.

[2]Jednotka rasy Zerg, které zvyšuje populační limit.



Obrázek 4.8: Hard



Obrázek 4.9: Hardest

Kapitola 5

Závěr

Pro splnění podcílů bylo nutné seznámit se a porozumět technikám a strategiím RTS her. Mnou navržené řešení agenta využívá několik instrukcí, které mají definováný postup, jaké akce se mají postupně provést. Tímto agent operuje na strategické úrovni, kde se učí používat tyto instrukce optimálně, aby mohl vytvořit co nejrychleji a nejefektivněji vojsko, které bude dostatečné pro poražení nepřítele.

Zprovoznění celého systému bylo z počátku velice komplikované. Framework ještě není zcela kompletní a tak nastávaly problémy jak ve spuštění samotné hry, tak se spuštěním agenta.

Po úspěšném nastavení všeho potřebného, jsem se již mohl pustit do prvotních implementací. Klíčové byly statické instrukce, které musely pokrývat základ, aby se měl agent od čeho odrazit. Nakonec jsem vybral pouze 2 instrukce, kde první určovala pozici, kde se budou přesouvat útočné jednotky po vytrénování a druhá byla pro budování budovy pro cvičení útočných jednotek. Všechny další instrukce již agent vybírá sám.

Agent má z pozorování několik jednoduchých hodnot, jako celková velikost populace, pozici kamery na mapě, populační limit a počet vytěžených surovin. Tyto informace používá pro generování Q-table dvojic. Díky těmto informacím se agent snaží vybírat co nejoptimálnější a nejlepší možné kombinace instrukcí.

Agent sám má definovanou množinu instrukcí, které může používat. Podle vybrané instrukce je poté agent ohodnocen. Pro implementaci tohoto agenta byl zvolen framework *PySC2*, který je zaměřený na strojové učení ve hře *Starcraft II* a poskytuje agentovi přesně takové informace, jaké má běžný hráč. Agent v porovnání s jinými implementacemi si vedl nejlépe. Dokázal se sám naučit, jakou posloupností instrukcí docílí úspěchu a zároveň dokázal najít místo, kde se nachází nepřátelská základna. Tím předčil všechna moje očekávání a splnil můj hlavní cíl, kterým bylo vytvořit agenta, který dokáže alespoň několikrát vyhrát. Pro implementaci svého agenta jsem vybral verzi hry *3.16.1*, která je první verzí, kterou framework *PySC2* rozjede. Na této verzi je postavená celá implementace agenta.

Mnou vytvořený agent není konečný. Agenta lze rozšiřovat jak o další instrukce, tak i o možnost stavění různých a pokročilých budov. Díky implementaci vyhledávacích sítí lze najít a označit jakoukoli budovu. Agent je implementován tak genericky, že po jednoduchých úpravách lze agenta naučit hrát i za zbylé rasy.

Literatura

[1] *Umělá inteligence - Význam*, [online]. 2016 [cit. 2018-05-2].
URL `https://it-slovnik.cz/pojem/umela-inteligence`

[2] *Markov decision process*, [online]. [cit. 2018-05-3].
URL `https://en.wikipedia.org/wiki/Markov_decision_process`

[3] *The Basis of Q-Learning*, [online]. [cit. 2018-05-4].
URL
`https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol2/zah/article2.html`

[4] Brownlee, J.: *Supervised and Unsupervised Machine Learning Algorithms*, [online]. 2016 [cit. 2018-05-2].
URL `https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/`

[5] Goldberg, D. E.; Holland, J. H.: Genetic algorithms and machine learning. *Machine learning*, ročník 3, č. 2, 1988: s. 95–99.

[6] Hastie, T.; Tibshirani, R.; Friedman, J.: Unsupervised learning. In *The elements of statistical learning*, Springer, 2009, s. 485–585.

[7] Hofmann, T.: Unsupervised learning by probabilistic latent semantic analysis. *Machine learning*, ročník 42, č. 1-2, 2001: s. 177–196.

[8] Hu, J.: *Reinforcement learning explained*, [online]. 2016 [cit. 2018-05-6].
URL `https://www.oreilly.com/ideas/reinforcement-learning-explained`

[9] Jaakkola, T.; Singh, S. P.; Jordan, M. I.: Reinforcement learning algorithm for partially observable Markov decision problems. In *Advances in neural information processing systems*, 1995, s. 345–352.

[10] Kotsiantis, S. B.; Zaharakis, I.; Pintelas, P.: Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, ročník 160, 2007: s. 3–24.

[11] Nasrabadi, N. M.: Pattern recognition and machine learning. *Journal of electronic imaging*, ročník 16, č. 4, 2007: str. 049901.

[12] Rábová, Z.; Hanáček, P.; Peringer, P.; aj.: *Užitečné rady pro psaní odborného textu*. FIT VUT v Brně, Listopad 2008, [Online; navštíveno 12.05.2015].
URL `http://www.fit.vutbr.cz/info/statnice/psani_textu.html`

[13] Sebastiani, F.: Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, ročník 34, č. 1, 2002: s. 1–47.

[14] Sutton, R. S.: Introduction: The challenge of reinforcement learning. In *Reinforcement Learning*, Springer, 1992, s. 1–3.

[15] Sutton, R. S.; Barto, A. G.: *Reinforcement learning: An introduction*, ročník 1. MIT press Cambridge, 1998.

[16] Watkins, C. J.; Dayan, P.: Q-learning. *Machine learning*, ročník 8, č. 3-4, 1992: s. 279–292.

Příloha A

Přiložené CD se zdrojovými kódy programu